Introduction to Reinforcement Learning

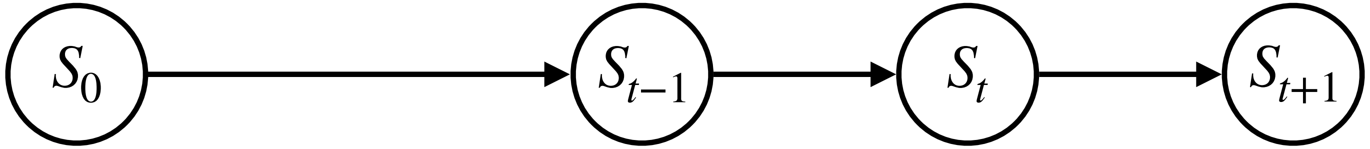
Lecture 6. Summary

Sungjoon Choi, Korea University

Markov Process

- A **Markov chain** is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event (aka Markov property).

- Random variable: S_t (state)



Markov Reward Process

- Now, we obtain rewards as we move to states.

$$S_0 \rightarrow R_1 \rightarrow S_1 \rightarrow R_2 \rightarrow S_2$$

- We have two random variables.
  - State: S_t
  - Reward: R_t
- Since the reward is a random variable, we take expectation to compute the reward function.
  - Reward function: $r(s) = \mathbb{E}[R_{t+1} \,|\, S_t = s]$

Markov Decision Process

ROBOT INTELLIGENCE LAB

- Formally, an MDP is a tuple (S, A, P, R, d):
  - A set of states $s \in S$.
  - A set of actions $a \in A$
  - A state transition function (or matrix)
    - $P(s'|s, a) = P(S_{t=1} = s|S_t = s, A_t = a)$
    - $P_{sas'} = P(s'|s, a)$
  - A reward function
    - $r(s) = \mathbb{E}[R_{t+1}|S_t = s]$
    - It depends on both state and action.
  - An initial state distribution d

Summary

ROBOT INTELLIGENCE LAB

- Bellman Equation

$$V_\pi(s) = \sum_a \pi(a \mid s) Q^{(}s, a)$$

$$Q_\pi(s, a) = \sum_{s'} \left[r(s, a, s') + \gamma V_\pi(s') \right] P(s' \mid s, a)$$

$$V_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} \left[r(s, a, s') + \gamma V_\pi(s') \right] P(s' \mid s, a)$$

$$Q_\pi(s, a) = \sum_{s'} \left[r(s, a, s') + \gamma \sum_{a'} Q_\pi(s', a') \pi(a' \mid s') \right] P(s' \mid s, a)$$

- Bellman **Optimality** Equation

$$V^*(s) = \max_a Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s'} \left[r(s, a, s') + \gamma V^*(s') \right] P(s' \mid s, a)$$

$$V^*(s) = \max_a \sum_{s'} \left[r(s, a, s') + \gamma V^*(s') \right] P(s' \mid s, a)$$

$$Q^*(s, a) = \sum_{s'} \left[r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right] P(s' \mid s, a)$$

$$\pi^*(a \mid s) = \arg\max_{a'} Q^*(s, a')$$

Q-Value Iteration

ROBOT INTELLIGENCE LAB

- Start from the random initial V_0
- For all states $s \in S$:

$$Q_k(s,a) = \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s' \mid s,a)$$

$$V_{k+1}(s) = \max_{a'} Q_k(s,a')$$

- We now have an explicit form of the policy:

$$\pi_{k=1}(a \mid s) = 1 \text{ for } a = \arg\max_{a'} Q_k(s,a')$$

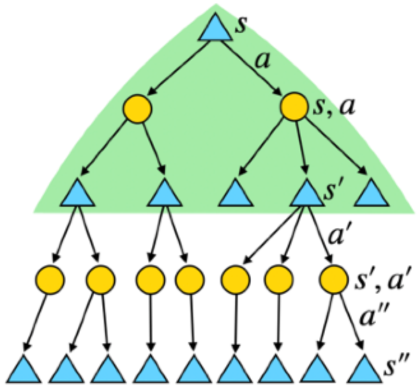
  - Note that this policy is **deterministic**.

Policy Iteration

- **Step 1: Policy evaluation**: Evaluate V_π.
- **Step 2: Policy improvement**: Generate π' where $V_{\pi'} \geq V_\pi$.

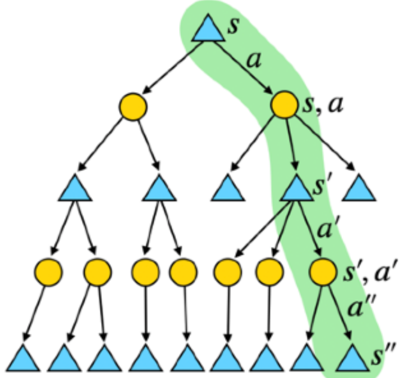
- **Policy iteration** is often more effective that **value iteration**, why?
  - It if often the case that a policy function reaches the optimal policy (policy iteration) much sooner than a value function reaches the optimal value function (value iteration).
  - In many cases, we are more interested in finding the optimal policy function rather than the optimal value function.

DP vs. MC vs. TD vs. SARSA



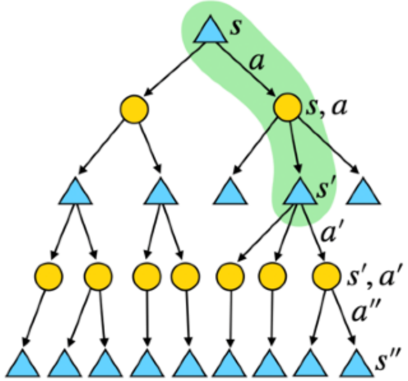
Dynamic Programming

$$V_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s'} \left[r(s,a,s') + \gamma V_k(s') \right] P(s' \mid s,a)$$



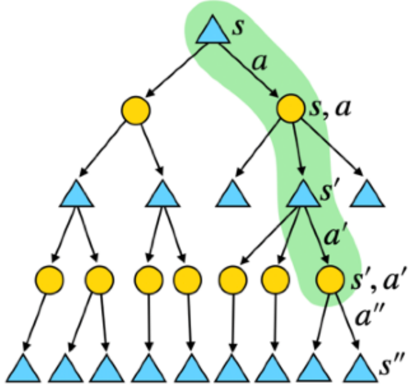
Monte-Carlo Learning

$$V_{k+1}(s) = V_k(s) + \alpha \left(G_t - V_k(s) \right)$$



Temporal Difference Learning

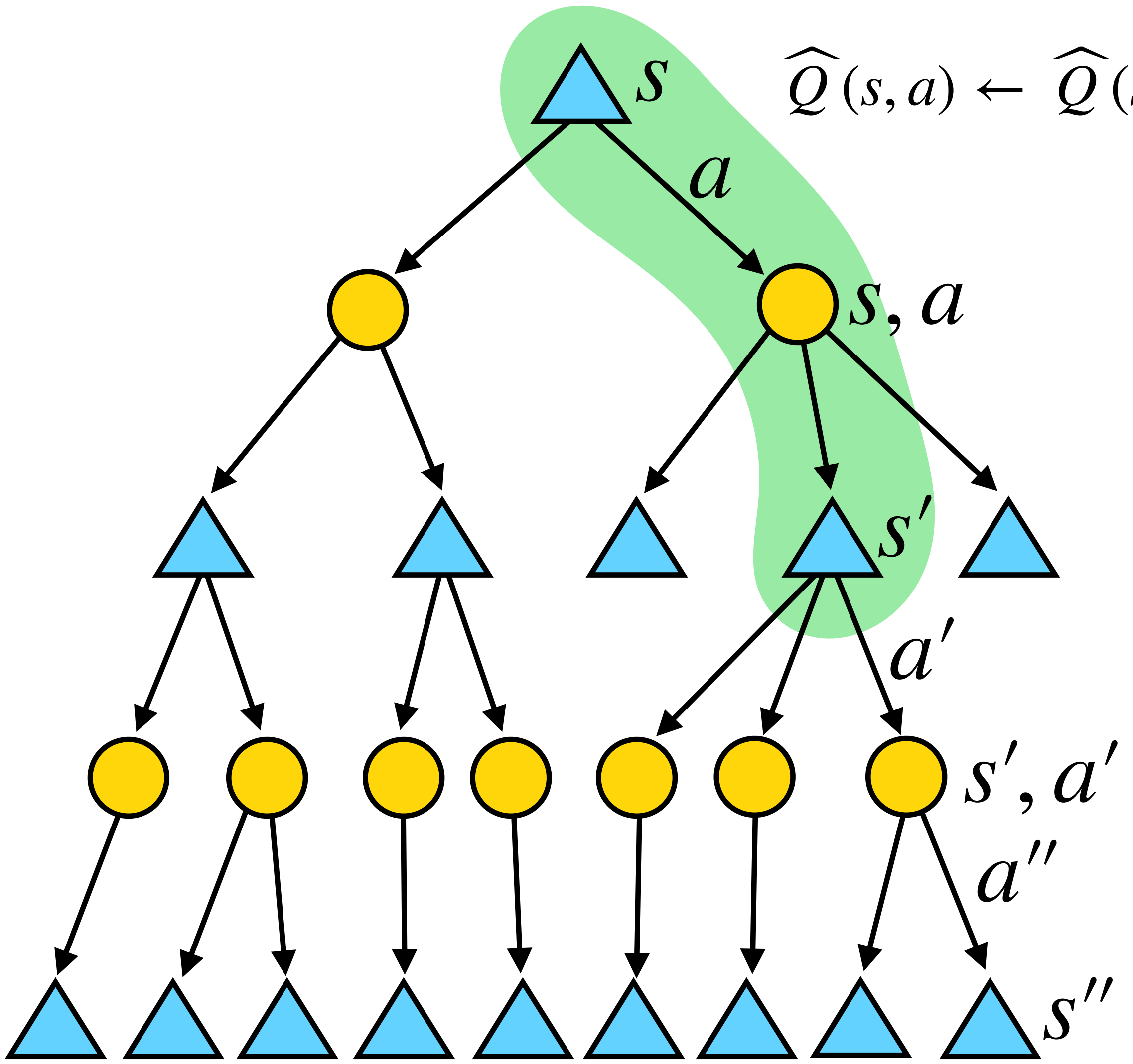
$$V_{k+1}(s) = V_k(s) + \alpha \left(r(s,a,s') + \gamma V_k(s') - V_k(s) \right)$$



SARSA

$$\widehat{Q}(s_t, a_t) \leftarrow \widehat{Q}(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \widehat{Q}(s_{t+1}, a_{t+1}) - \widehat{Q}(s_t, a_t) \right)$$

Q-Learning



$$\widehat{Q}(s,a) \leftarrow \widehat{Q}(s,a) + \alpha\left(r(s,a,s') + \gamma \max_{a'} \widehat{Q}(s',a') - \widehat{Q}(s,a)\right)$$

On-Policy vs. Off-Policy Learning

ROBOT INTELLIGENCE LAB

- On-Policy Learning (MC, SARSA)
  - Learn the value of the policy π using the episodes sampled from π.
- Off-Policy Learning
  - Learn the value of the policy π using the episodes sampled from **any arbitrary** μ.

- Why is off-policy learning important?
  - Off-policy methods enable learning from observing humans or other agents.
  - Re-use experiences generated from old polices (**experience replay**).
  - Learn the optimal policy while following other exploratory policy.

Summary

Markov Decision Process

Policy Evaluation

$$V_{k+1}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} \left[r(s,a,s') + \gamma V_k(s')\right] P(s'|s,a)$$

Q-Policy Iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} \left[r(s,a,s') + \gamma \sum_{a'} Q_k(s',a')\pi(a'|s')\right] P(s'|s,a)$$

Q-Value Iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} \left[r(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right] P(s'|s,a)$$

Model-Free Methods

TD Learning

$$V_{k+1}(s) \leftarrow V_{k+1}(s) + \alpha \left(r(s,a,s') + \gamma V_k(s') - V_k(s)\right)$$

SARSA

$$Q_{k+1}(a,a) \leftarrow Q_k(s,a) + \alpha \left(r(s,a,s') + \gamma Q_k(s',a') - Q_k(s,a)\right)$$

Q-Learning

$$Q_{k+1}(a,a) \leftarrow Q_k(s,a) + \alpha \left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)\right)$$

Deep Q Network (DQN)

- (Stable) Update Rule

$$L(\theta) = \sum_i \left(r_i + \gamma \max_{a'} Q(s_i', a'; \theta^-) - Q(s_i, a_i; \theta) \right)^2$$

- Delayed update
  - For numerical stability, slowly update the target network
  - θ^-: previous parameter (for the Q estimation)
  - θ: current parameter to update
- Other tricks
  - Gradient clipping
  - Input normalization

Double Deep Q Network (DDQN)

ROBOT INTELLIGENCE LAB

- Recall the original DQN loss function

$$L(\theta) = \sum_i \left(r_i + \gamma \max_{a'} Q(s_i', a'; \theta^-) - Q(s_i, a_i; \theta) \right)^2$$

- Note that the function for selecting the current optimal action and evaluating the optimal action is the same ,$Q(s, a; \theta^-)$.
- It often leads to **over-optimism** of the Q function.
- To resolve this issue,

$$L(\theta) = \sum_i \left(\underbrace{\boxed{r_i + \gamma Q(s_i', \boxed{\arg\max_{a'} Q(s_i', a'; \theta)}; \theta^-)}}_{\text{Target}} - \underbrace{\boxed{Q(s_i, a_i; \theta)}}_{\text{Prediction}} \right)^2$$

Prioritized Experience Replay (PER)

ROBOT INTELLIGENCE LAB

- The intuition is to give **more emphasis** on unfitted data.
- Implementation is simple:
  - Sample k transitions from the experience replay from $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$

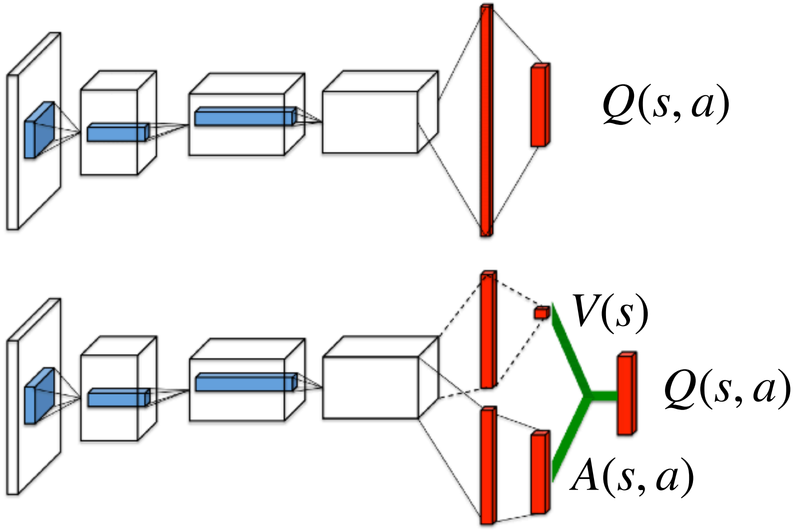
$$\text{where } p_i = \left| r_i + \gamma Q(s_i', \arg\max_{a'} Q(s_i', a'; \theta); \theta^-) - Q(s_i, a_i; \theta) \right|.$$

  - The (weighted) update rule is:

$$L(\theta) = \sum_i w_i \left(r_i + \gamma Q(s_i', \arg\max_{a'} Q(s_i', a'; \theta); \theta^-) - Q(s_i, a_i; \theta) \right)^2$$

$$\text{where } w_i = \left(\frac{1}{n \cdot p_i} \right)^\beta / \max(w).$$

Dueling Architecture



- The main idea is to separate $Q(s, a)$ into $V(s)$ and $A(s, a)$ which is the advantage.
- Advantage function
  - $Q_\pi(s, a) = V_\pi(s) + A_\pi(s, a)$
  - $\arg\max_{a'} A(s, a') = \arg\max_{a'} Q(s, a')$

How to compute the gradients

$$\nabla_\theta \eta(\pi_\theta) = \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid \pi_\theta\right]$$

- Policy Gradient Theorem:

$$\nabla_\theta \eta(\pi_\theta) = \frac{1}{(1-\gamma)} \sum_s \rho_{\pi_\theta} \sum_a \nabla_\theta \pi_\theta(a \mid s) Q^{\pi_\theta}(s, a)$$

$$\nabla_\theta \eta(\pi_\theta) \approx \nabla_\theta \log \pi_\theta(a_t \mid s_t) Q_{\pi_\theta}(s_t, a_t)$$

- Note that we only require the gradient of $\pi_\theta(\cdot)$ not $Q^{\pi_\theta}(\cdot)$!

R. Sutton et al, "Policy Gradient Methods for Reinforcement Learning with Function Approximation", NeurIPS, 2000

Trust Region Policy Optimization

ROBOT INTELLIGENCE LAB

$$\max_{\theta_{i+1}} L_{\pi_{\theta_i}}(\pi_{\theta_{i+1}}) = \mathbb{E}_{s\sim\rho_{\pi_{\theta_i}}, a\sim\pi_{\theta_i}}\left[\frac{\pi_{\theta_{i+1}}(a\,|\,s)}{\pi_{\theta_i}(a\,|\,s)} A_{\pi_{\theta_i}}(s,a)\right]$$

$$\text{subject to } D_{KL}^{\rho}(\pi_{\theta}, \pi_{\theta_{i+1}}) \leq \delta$$

- In summary,
  - TRPO is a minorization maximization framework for RL.
  - Interpretation of the trust region method:
    1. Update policy distribution slowly
    2. Consider the geometry of the distribution space
  - There are two approximations: 1) $\mathbb{E}_{s\sim\rho_{\pi'}} \Rightarrow \mathbb{E}_{s\sim\rho_{\pi}}$ and 2) $D_{KL}^{\max} \Rightarrow D_{KL}^{\rho}$

Proximal Policy Optimization (Adaptive KL Penalty)

ROBOT INTELLIGENCE LAB

- The TRPO objective is:

$$\max_{\theta} \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] \text{ s.t. } D^{\rho}_{KL}\left[\pi_{old}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right] \le \delta$$

- The unconstrained objective of TRPO is:

$$L(\theta) = \max_{\theta} \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t - \beta D^{\rho}_{KL}\left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right]$$

- The proposed adaptive KL penalty method is to adaptively change β by checking $d = \mathbb{E}_t\left[D_{KL}[\pi_{\theta_{old}}, \pi_\theta]\right]$:
  - If $d < d_{targ}/1.5$, $\beta \leftarrow \beta/2$
  - If $d > d_{targ} \times 1.5$, $\beta \leftarrow \beta \times 2$

Soft Actor-Critic

ROBOT INTELLIGENCE LAB

- SAC learns three functions: $V_\psi(s)$, $Q_\theta(s, a)$, and $\pi_\phi(a \mid s)$.
- For learning $V_\psi(s)$:

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\frac{1}{2}\left(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}\left[Q_\theta(s_t, a_t) - \log \pi_\phi(a_t \mid s_t)\right]\right)^2\right]$$

where actions are being sampled from the current policy $\pi_\phi(a \mid s)$ not from the replay.

- For learning $Q_\theta(s, a)$:

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t)\right)^2\right] \text{ where } \hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1}}\left[V_\psi(s_{t+1})\right]$$

- For learning $\pi_\phi(a \mid s)$:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[D_{KL}\left(\pi_\phi(\cdot \mid s_t) \| \frac{\exp\left(Q_\theta(s_t, \cdot)\right)}{Z_\theta(s_t)}\right)\right]$$

If we reparameterize the stochastic policy $a_t = f_\phi(\epsilon_t; s_t)$ where ϵ_t is sampled from some distribution,

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D, \epsilon_t \sim \mathcal{N}}\left[\log \pi_\phi\left(f_\phi(\epsilon_t; s_t) \mid s_t\right) - Q_\theta\left(s_t, f_\phi(\epsilon_t; s_t)\right)\right]$$

Augmented Random Search

ROBOT INTELLIGENCE LAB

Algorithm 2 Augmented Random Search (ARS): four versions **V1**, **V1-t**, **V2** and **V2-t**

1: **Hyperparameters:** step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν, number of top-performing directions to use b ($b < N$ is allowed only for **V1-t** and **V2-t**)

Use top b search directions.

2: **Initialize:** $M_0 = \mathbf{0} \in \mathbb{R}^{p \times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n \times n}$, $j = 0$.

3: **while** ending condition not satisfied **do**

4: Sample $\delta_1, \delta_2, \ldots, \delta_N$ in $\mathbb{R}^{p \times n}$ with i.i.d. standard normal entries.

5: Collect $2N$ rollouts of horizon H and their corresponding rewards using the $2N$ policies

$$\mathbf{V1:} \quad \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$$

$$\mathbf{V2:} \quad \begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\, \mathrm{diag}\,(\Sigma_j)^{-1/2}\,(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\, \mathrm{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$$

Input normalization

for $k \in \{1, 2, \ldots, N\}$.

6: Sort the directions δ_k by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$ denote by $\delta_{(k)}$ the k-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.

7: Make the update step:

$$M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)},$$

where σ_R is the standard deviation of the $2b$ rewards used in the update step.

8: **V2** : Set μ_{j+1}, Σ_{j+1} to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.[2]

9: $j \leftarrow j + 1$

10: **end while**

.

Thank You

ROBOT INTELLIGENCE LAB